Benchmarking UAQ Solvers

Alessandro Armando
alessandro.armando@unige.it
DIBRIS, Università di Genova
Genova, Italia

Giorgia A. Gazzarata
giorgia.gazzarata@unige.it
DIBRIS, Università di Genova
Genova, Italia

Fatih Turkmen
f.turkmen@rug.nl
University of Groningen
Groningen, Netherlands

ABSTRACT

The User Authorization Query (UAQ) Problem is key for RBAC systems that aim to offer permission level user-system interaction, where the system automatically determines the roles to activate in order to enable the requested permissions. Finding a solution to a UAQ problem amounts to determining an optimum set of roles to activate in a given session so to obtain some permissions while satisfying a collection of authorization constraints, most notably Dynamic Mutually-Exclusive Roles (DMER) constraints. Although the UAQ Problem is NP-hard, a number of techniques to solve the UAQ problem have been put forward along with encouraging, albeit inconclusive, experimental results. We propose a methodology for designing parametric benchmarks for the UAQ problem and make a novel suite of parametric benchmarks publicly available that allows for the systematic assessment of UAQ solvers over a number of relevant dimensions. By running three prominent UAQ solvers against our benchmarks, we provide a very comprehensive analysis showing *(i)* the shortcomings of currently available benchmarks, *(ii)* the adequacy of the proposed methodology and *(iii)* that the reduction to PMaxSAT is currently the most effective approach to tackling the UAQ problem.

1 INTRODUCTION

The User Authorization Query (UAQ) Problem [5] amounts to determining an optimum set of roles to activate in a given session in order to obtain some permissions while satisfying a collection of authorization constraints, most notably Dynamic Mutually-Exclusive Roles (DMER) constraints.

The UAQ problem is key to support *permission level* user-system interaction (where the system automatically determines the roles that need to be activated in order to enable the requested permissions), as opposed to *role level* interaction (where it is the user who explicitly determines and tells the system which roles must be activated). UAQ solvers can thus help to improve the usability of most advanced access management and enforcement systems (e.g. Apache Fortress[1]) offering full support to the RBAC standard [15] by relieving the user from the burden of dynamically choosing the roles to activate for the task at hand. This choice could indeed be complex as the set of roles to activate must *(i)* enable the execution of desired functionalities, *(ii)* comply with the authorization constraints prescribed by the security policy and, e.g., *(iii)* minimize the activation of unnecessary permissions (which represent a risk). UAQ solvers can also play an important role by offering improved support to the enforcement of RBAC policies in microservices architectures.[2] In this context, determining what is the most appropriate set of roles to activate for a given service is not only difficult but must be done very quickly.

In a UAQ problem the permissions requested by the user come in two sets: a lower bound P_{lb} and an upper bound P_{ub} such that $P_{lb} \subseteq P_{ub} \subseteq P$, where P is the complete set of permissions. The permissions in P_{lb} are those that *must* be granted, whereas those in $P_{ub} \setminus P_{lb}$ are additional permissions that *can* be granted. It is then possible to either minimize or maximize the number of permissions in $P_{ub} \setminus P_{lb}$ to be granted depending on which objective (safety or availability, respectively) has priority. If safety (availability) is chosen, then the number of permissions from $P_{ub} \setminus P_{lb}$ needs to be minimized (maximized, resp.). Notice that a certain degree of safety is achieved even if availability is preferred over safety, since no permission in $P \setminus P_{ub}$ can be granted. DMER constraints are of the form $DMER(\{r_1, \ldots, r_m\}, t)$. They constrain activation of roles by requiring that no user can activate t or more roles in $\{r_1, r_2, \ldots, r_m\}$.

Although the UAQ Problem is NP-hard [4, 5], a number of techniques to solve the UAQ problem have been put forward along with encouraging experimental results. These approaches can be broadly classified in two classes: *search-based techniques* [13, 18], whereby the problem is solved by systematically exploring a suitably defined search space, and *SAT-based techniques* [2, 12, 13, 18], i.e. techniques that leverage an encoding of the problem into either *(i)* a sequence of propositional satisfiability problems (SAT) [12, 13, 18] or *(ii)* a partial maximum propositional satisfiability problem (PMaxSAT) [2, 18] and then use SAT solvers and PMaxSAT solvers respectively as workhorses to find a solution to the problem (if any). SAT-based approaches leverages the fast paced advancements achieved by lively and dedicated research communities that organize competitions[3] and evaluations[4] of state-of-the-art solvers on a yearly basis.

Most of the techniques proposed in the literature have been experimentally evaluated by running them against different benchmark problems. These benchmarks are usually parametric in some relevant dimension of the problem (e.g. number of roles, number of DMER constraints, number of requested permissions) and aim at evaluating the scalability of the proposed techniques along them. The current state of affairs is nevertheless unsatisfactory for a number of reasons. The available benchmarks do not cover (and thus do not test the solvers against) all the relevant aspects of the problem. For instance, the problems used in [13] do not consider the case where the number of roles to activate is maximized (i.e. $obj = max$) and therefore do not allow the evaluation of UAQ solvers in case of $obj = max$. Furthermore, solvers are often evaluated against problems proposed by the same authors and this does not permit to assess the relative merits of the proposed techniques.

[1]https://directory.apache.org/fortress
[2]https://kubernetes.io/docs/reference/access-authn-authz/rbac/
[3]http://www.satcompetition.org
[4]https://maxsat-evaluations.github.io/

By leveraging the asymptotic complexity analysis results given in [13], in this paper we propose a methodology for designing parametric benchmarks for the UAQ problem. As we will see, our methodology leads to benchmarks capable to *(i)* stress test solvers along dimensions of the problem for which no polynomial-time technique is known but also *(ii)* to check their effectiveness, by determining whether they efficiently solve problems that are known to be solvable in polynomial time.

By using our methodology we introduce and make publicly available a novel suite of parametric benchmarks that allows for the systematic assessment of UAQ solvers over a number of relevant dimensions. These include problems for which no polynomial-time algorithm is known as well as problems for which polynomial-time algorithms do exist. For each benchmark we indicate its purpose and the expected behavior of solvers. The suite consists of 23 parametric benchmarks grouped in 14 families: 11 benchmarks (from 7 families) have the safety objective and 12 (from 7 families) have the availability objective. For each benchmark and each considered value of the parameter the suite includes 10 different problem instances. This contribution is particularly important. In fact, the lack of adequate benchmarks makes it difficult to assess the efficiency of the different techniques used to tackle the UAQ problem. Even more so, it makes it difficult to compare their performance. By providing a complete and adequate suite of benchmarks, we are de facto offering the possibility to both assess the performance of a single solver, and compare the performance of the different solutions. [6] provided a similar contribution for the famous Symmetric and Asymmetric Traveling Salesperson Problems. This work helped to *(i)* reduce the publication of relatively weak TSP heuristics and *(ii)* motivate development of new TSP approaches.

We have used the new suite of benchmark problems as well as the benchmarks introduced in [13] to experimentally evaluate three state-of-the-art solvers: 2D-Opt-Search [13] a search-based solver, 2D-Opt-CNF [13] that combines the reduction of the UAQ Decision Problem to SAT, a state-of-the-art SAT solver and a binary search, and AQUA [2], a SAT-based solver that implements a reduction of the UAQ Problem to PMaxSAT and uses any state-of-the-art PMaxSAT solver to tackle the problem. The experiments provide a comprehensive and comparative analysis of unprecedented breadth from which it is possible to draw a number observations:

- 2D-Opt-CNF and AQUA quickly solve all benchmark problems taken from [13]. This result indicates that this suite of benchmarks does not represent adequately the complexity of the problem and it is therefore of limited use to assess the effectiveness of the solvers.
- 2D-Opt-CNF and 2D-Opt-Search scale poorly for (most) benchmarks in our suite for which polynomial-time algorithms exist. This results is in stark contrast with the experimental results given in [13] from which one could be led to believe that these two solvers scale well in practice.
- For most benchmarks (22 out of 23), the behavior of AQUA meets the expectations that can be drawn from the known asymptotic complexity analysis results; this results is a strong indication of the validity of the methodology we propose in this paper, but it also indicates that there is still room for further investigation.
- AQUA outperforms both 2D-Opt-Search and 2D-Opt-CNF in the vast majority of the benchmark considered. This is hardly surprising since PMaxSAT solver implements sophisticated search algorithms specifically tailored to tackle optimization problems; in contrast, 2D-Opt-CNF tackles the optimization problem through a fairly naive binary search strategy and 2D-Opt-Search simply enumerates the solutions in order to find an optimum. Yet, prior to our experimental analysis, there was little empirical evidence (if any) to support this conclusion.
- We single out a benchmark, namely Pub, which can be solved efficiently but that all solvers we considered in our experiments find it difficult to tackle. We believe that further analysis on this family of problems could lead to improvements of existing solvers.

Structure of the paper. In the next section we introduce the UAQ problem. In Section 3 we provide an overview of the techniques for solving the problem. In Section 4 we present our methodology for the generation of parametric benchmark problems and introduce the new suite of benchmarks. In Section 5 we present and discuss the experimental results. In Section 6 we discuss the related work and in Section 7 we conclude the paper with some final remarks.

2 THE UAQ PROBLEM

An *RBAC policy* is a tuple $RP = (U, R, P, UA, PA, \geq, C)$, where U is a set of users, R a set of roles, and P a set of permissions; users are associated to roles by the *user-assignment* relation $UA \subseteq U \times R$ and roles are associated to permissions by *permission-assignment* relation $PA \subseteq R \times P$; $\geq$ is a partial order on R, modeling the hierarchy between roles, i.e. $r_1 \geq r_2$ means that r_1 is *more senior than* (has all permissions of) r_2 for $r_1, r_2 \in R$; and C is a set of *dynamic mutually exclusive role (DMER)* constraints of the form $\text{DMER}(\{r_1, \ldots, r_m\}, t)$, with $t \leq m$, stating that no user can simultaneously activate t or more roles in the set $\{r_1, \ldots, r_m\}$.

Let $u \in U$, we define $R_u = \{r' \in R : r \geq r'$ for some $r \in R$ such that $(u, r) \in UA\}$. A user u *has permission* p iff $(r, p) \in PA$ for some $r \in R_u$. We assume that RBAC policies considered are *finite*, i.e. U, R, and P have finite cardinality. Moreover, we treat permissions as if they are opaque (i.e. we do not consider the internal structure of permissions) and mutually independent, i.e. the possession of one or more permissions does not imply the possession of another permission. Let $p \in P$, we define $R_p = \{r' \in R : r' \geq r$ for some $r \in R$ such that $(r, p) \in PA\}$ and $R_{P'} = \bigcup_{p \in P'} R_p$ for any $P' \subseteq P$. Similarly, if $r \in R$ then $P_r = \{p \in P : (r', p) \in PA$ for some $r' \in R$ such that $r \geq r'\}$ and $P_{R'} = \bigcup_{r \in R'} P_r$ for any $R' \subseteq R$. Let $\varrho \subseteq R$, then we say that ϱ *satisfies* $\text{DMER}(\{r_1, \ldots, r_m\}, n)$ iff $|\{r_1, \ldots, r_m\} \cap \varrho| < n$ and that ϱ *satisfies* C iff ϱ satisfies c for all $c \in C$.

Let S be a set of sessions and $user : S \to U$ a function that associates each session $s \in S$ with the corresponding user.

A *User Authorization Query (UAQ)* is a tuple $q = (s, P_{lb}, P_{ub}, obj)$, where $s \in S$, $P_{lb} \subseteq P_{ub} \subseteq P$, and $obj \in \{\text{any}, \text{min}, \text{max}\}$.

Definition 2.1 (UAQ Problem). The *UAQ Problem for* $q = (s, P_{lb}, P_{ub}, obj)$ *in RP* is the problem of determining a set of roles $\varrho \subseteq R_{user(s)}$ such that *(i)* ϱ satisfies C, *(ii)* $P_{lb} \subseteq P_\varrho \subseteq P_{ub}$, and *(iii)* any other $\varrho' \subseteq R_{user(s)}$ that satisfies C and $P_{lb} \subseteq P_{\varrho'} \subseteq P_{ub}$ is such that

- $P_\varrho \subseteq P_{\varrho'}$, if $obj = \text{min}$;

- $P_{\varrho'} \subseteq P_{\varrho}$, if $obj = \max$.

In case obj = any, any ϱ satisfying the *(i)* and *(ii)* is a valid solution.

Definition 2.2 (UAQ Decision Problem). Let $w = (s, P_{lb}, P_{ub}, k_p)$ where $k_p \in \{\leq, \geq\} \times [0, |P_{ub} \setminus P_{lb}|]$. The *UAQ Decision Problem for w in RP* is the problem of determining a set of roles $\varrho \subseteq R_{user(s)}$ such that *(i)* ϱ satisfies C, *(ii)* $P_{lb} \subseteq P_{\varrho} \subseteq P_{ub}$, and

(iii.a) $|P_{\varrho} \setminus P_{lb}| \leq n$ if $k_p = (\leq, n)$,
(iii.b) $|P_{\varrho} \setminus P_{lb}| \geq n$ if $k_p = (\geq, n)$.

For the sake of simplicity but without loss of generality, in our work we assume that no role in R is assigned permissions that are not contained in P_{ub}, and that $P_{ub} = P$. In fact, all the permissions in $P \setminus Pub$ *must not be granted*; therefore all the roles that activate these permissions *cannot be included in a solution*. These roles and the permissions in $P \setminus P_{ub}$ can thus be safely removed from the policy (in polynomial time). By doing that, we simply exclude UAQ problems that are de facto equivalent to UAQ problems that meet our assumptions.

3 SOLVING THE UAQ PROBLEM

In this section, we provide a systematic overview of search-based and SAT-based techniques for solving the UAQ problem. This will also give insights into the complexity of the problem.

3.1 Search-based Techniques

A naive two-step algorithm that first obtains a set of roles covering the desired permissions minimally and then checks whether the set of roles satisfies the constraints is proposed in [20]. However, the algorithm may not find a combination of roles that satisfies the constraints since the first step does not take any constraints into account.

An alternative approach, discussed again in [20] and improved in [18], amounts to

Algorithm 1

(1) enumerating all possible role activations for the user,
(2) checking (in polynomial time as shown in [13]) whether the selected roles grant the requested permissions (i.e. fall between P_{lb} and P_{ub}), and satisfy the DMER constraints, and
(3) keeping the best (according to the security objective considered) solution encountered, if any.

The algorithm is clearly in $O(2^{|R|})$. The DPPL-based algorithms proposed in [18] and the DFS-based algorithms proposed in [9] are optimized versions of this algorithm with additional preprocessing and pruning steps. An alternative approach to solving the UAQ problem (adapted from [13]) is as follows:

Algorithm 2

(1) enumerate all sets of roles $S_p \subseteq R_p$ for each $p \in P_{ub}$,
(2) check in polynomial time whether $S = \bigcup_{p \in P_{ub}} S_p$ is such that $P_{lb} \subseteq P_S \subseteq P_{ub}$ and S satisfies the DMER constraints, and
(3) keep the best (according to the security objective considered) solution encountered, if any.

The above algorithm is in $O(2^{\widehat{R_P}|P_{ub}|})$, where $\widehat{R_P} = \max_{p \in P} |R_p|$. To see this it suffices to observe that for each $p \in P_{ub}$ there are at most $2^{|R_p|}$ subsets S_p of R_p and thus there are in total at most $2^{\widehat{R_P}|P_{ub}|}$ candidate solutions to consider. When it is sufficient to activate "at most one role per permission", [13] shows that the above algorithm is in $O(\widehat{R_P}^{|P_{ub}|})$ and hence is also fixed-parameter polynomial (FPP) in $|P_{ub}|$, i.e. it is polynomial-time if $|P_{ub}| \leq c$ for some constant c.

If the objective is min or any, then it is sufficient to activate at most one role per permission and this leads to the following, more efficient version of the algorithm:

Algorithm 3

(1) enumerate all roles $r_p \in R_p$ for each $p \in P_{lb}$,
(2) check in polynomial time whether $S = \{r_p \in R_p : p \in P_{lb}\}$ is such that $P_S \subseteq P_{ub}$ and S satisfies the DMER constraints, and
(3) keep the best (according to the security objective considered) solution encountered, if any.

If the objective is min or any, it is in fact possible to consider the activation of individual roles granting the permissions in P_{lb} and the algorithm is in $O(|\widehat{R_P}|^{|P_{lb}|})$ [13].

We note that Algorithm 3 (asymptotically) outperforms Algorithm 2 if $obj = min$. To see this, it suffices to note that $2^{\widehat{R_P}|P_{ub}|} > \widehat{R_P}^{|P_{lb}|}$ implies $\widehat{R_P}|P_{ub}| > \log_2 \widehat{R_P}^{|P_{lb}|}$ which in turn implies $|P_{ub}| > |P_{lb}| \frac{\log_2 \widehat{R_P}}{\widehat{R_P}}$. This is always true since $P_{lb} \subseteq P_{ub}$ (and consequently $|P_{lb}| \leq |P_{ub}|$) and $\frac{\log_2 x}{x} < 1$ for all x.

If the optimization objective is max, then the "at most one role per permission" assumption does not hold and hence the FPP result cannot be applied in general. To illustrate consider a UAQ problem with $R = \{r_1, r_2, r_3\}$, $P = \{p_1, p_2, p_3, p_4\}$, PA such that $P_{r_1} = \{p_1, p_3\}$, $P_{r_2} = \{p_2, p_4\}$, $P_{r_3} = \{p_2, p_3\}$, $P_{lb} = \{p_1\}$, $P_{ub} = P$ and $obj = \max$. The solution $\{r_1\}$ satisfies the "at most one role per permission" assumption but it is not optimal. In fact both $\{r_1, r_2\}$ and $\{r_1, r_2, r_3\}$ activate a larger (actually maximal) set of permissions, namely P. An alternative approach to tackling the problem for the max case is put forward in [13]:

Algorithm 4

(1) enumerate all sets of possible role activations $R_a = R_1 \cup \cdots \cup R_n \cup R_{free}$, where $R_i \subseteq rs_i$ and $|R_i| < t_i$, for all constraints $DMER(rs_i, t_i)$ in C and $i = 1, \ldots, |C|$, and $R_{free} \subseteq R$ is the set roles that do not occur in the DMER constraints (and can thus be freely activated),
(2) check in polynomial time whether $P_{lb} \subseteq P_{R_a} \subseteq P_{ub}$ and R_a satisfies the DMER constraints.
(3) keep the maximum sized $|P_{R_a}|$ solution encountered, if any.

We now note that for each constraint $DMER(rs_i, t_i)$ in C there are $\sum_{k=1}^{t} \binom{|rs_i|}{k}$ subsets R_i of rs_i such that $|R_i| < t_i$. The number of sets $R_1 \cup \cdots \cup R_n$ is $\left(\sum_{k=1}^{t} \binom{|rs_i|}{k}\right)^{|C|}$, from which it easily follows that the enumeration of the set of roles $R_a = R_1 \cup \cdots \cup R_n \cup R_{free}$ grows as $O(\widehat{rs}^{|C|\widehat{t}})$, where $\widehat{rs} = \max_{DMER(rs,t) \in C} |rs|$ and $\widehat{t} = \max_{DMER(rs,t) \in C} t$. This improves the upper bound $O(|R|^{|C|\widehat{t}})$ given in [13]. As shown in [13], the role hierarchy, $\succeq$, does not contribute to the computational complexity of the problem.

Table 1: Complexity of solution techniques

Algorithm	Objective	Complexity
1	any, min, max	$O(2^{\|R\|})$
2	any, min, max	$O(2^{\widehat{R_P}\|P_{ub}\|})$
3	any, min	$O(\widehat{R_P}^{\|P_{lb}\|})$
4	max	$O(\widehat{rs}^{\|C\|\hat{t}})$

A summary of the results is given in Table 1

3.2 SAT-based Techniques

Let $RP = (U, R, P, UA, PA, \geq, C)$ be an RBAC policy with constraints and $q = (s, P_{lb}, P_{ub}, obj)$ a UAQ query for RP. Since RP is finite (i.e. the set U of users, R of roles, and P of permissions are all finite), the UAQ problem can be tackled by leveraging SAT solvers. This can be done in a variety of ways. A first approach [13] amounts to reducing the UAQ Decision Problem to SAT and solving the optimization problem through binary search that leverage the SAT solver as an oracle for the decision problem. The second approach ([2, 17, 18]) eliminates the need for the binary search by directly encoding UAQ problems into PMaxSAT.

3.2.1 Reducing the UAQ Decision Problem to SAT. Let $w = (s, P_{lb}, P_{ub}, k_p)$ where $k_p \in \{\leq, \geq\} \times [0, |P_{ub} \setminus P_{lb}|]$. The reduction of the UAQ Decision Problem for w in RP to SAT amounts to generating a set of clauses $C^w_{RP} = C_{RP} \cup C^q$, where C_{RP} and C^w are defined below. We assume the existence of a propositional variable $\overline{r}$ for each $r \in R$ and a propositional variable $\overline{p}$ for each $p \in P$.

C_{RP} is the smallest set of propositional clauses satisfying the following conditions.

Core RBAC.

(1) for all $r \in R$ if $(user(s), r) \notin UA$ then $\neg\overline{r} \in C_{RP}$;
(2) for all $p \in P$ and $r \in R$ such that $(p, r') \in PA$ with $r \geq r'$, $(\neg\overline{r} \vee \overline{p}) \in C_{RP}$;
(3) for all $p \in P$, $(\neg\overline{p} \vee \bigvee\{\overline{r} : \text{exists } r' \in R, r \geq r', (p, r') \in PA\}) \in C_{RP}$.

It is easy to see that the number of clauses above is in $O(|R||P|)$ and the number of propositional variables in $O(|R| + |P|)$.

DMER Constraints. For all $DMER(rs, n) \in C$, a CNF of the following formula is in C_{RP}:

$$\sum_{r \in rs} \overline{r} \leq n - 1 \tag{1}$$

As shown in [16], inequalities of the form $\sum_{x \in X} x \leq n$ can be succinctly encoded into CNF with $7|X|$ clauses and $2|X|$ additional propositional variables. Thus, constraints of the form (1) can be encoded with a number of variables and clauses in $O(|R|)$.

C^w is the smallest set of propositional clauses satisfying the following conditions.

Query.

- a unit clause $\overline{p} \in C_{RP}$ for each $p \in P_{lb}$;
- a unit clause $\neg\overline{p} \in C_{RP}$ for each $p \in P \setminus P_{ub}$;
- A CNF of the following formula in C_{RP}:

$$\sum_{p \in P_{ub} \setminus P_{lb}} \overline{p} \leq n \text{ if } k_p = \langle \leq, n \rangle$$
$$\sum_{p \in P_{ub} \setminus P_{lb}} \overline{p} \geq n \text{ if } k_p = \langle \geq, n \rangle$$

It can be shown that any solution to C^w_{RP} corresponds to a solution of the corresponding UAQ Decision Problem and vice versa.

3.2.2 Reducing the UAQ Problem to PMaxSAT. A PMaxSAT problem is given by a pair $\langle \mathcal{H}, \mathcal{S} \rangle$, where $\mathcal{H}$ and $\mathcal{S}$ are two finite sets of clauses, called "hard" and "soft" respectively. A solution to a PMaxSAT problem $\langle \mathcal{H}, \mathcal{S} \rangle$ is any truth-value assignment to the variables in $\mathcal{H}$ and $\mathcal{S}$ that satisfies all clauses in $\mathcal{H}$ and the maximum number of clauses in $\mathcal{S}$.

A UAQ Problem for $q = (s, P_{lb}, P_{ub}, obj)$ can be reduced to a PMaxSAT problem $\langle \mathcal{H}^q_{RP}, \mathcal{S}^q \rangle$, where $\mathcal{H}^q_{RP}$ is obtained from C_{RP} (as defined in Section 3.2.1) by adding

- a unit clause $\overline{p}$ for each $p \in P_{lb}$ and
- a unit clause $\neg\overline{p}$ for each $p \in P \setminus P_{ub}$;

and $\mathcal{S}^q$ comprises

- a unit clause $\neg\overline{p}$ if $obj = \min$
- a unit clause $\overline{p}$ if $obj = \max$

for each $p \in P_{ub} \setminus P_{lb}$. No soft clauses are included if $obj = $ any.

It can be shown that any solution to C^w_{RP} (C^q_{RP}) corresponds to a solution of the corresponding UAQ Decision Problem (UAQ Problem, resp.) and vice versa.

It can be readily seen that for both reductions the number of clauses is in $O(|R||P|)$ and the number of propositional variables is in $O(|R| + |P|)$.

4 BENCHMARKS

Designing benchmarks suitable for the systematic assessment of UAQ solvers is not easy. A common approach [2, 9, 13, 18] is to focus on families of problems that are parametric on some aspects of the problem that may contribute to its complexity. All other aspects are either set to a predefined, constant value or are randomly chosen in a given interval or according to some criterion. By running a solver against the instances corresponding to increasing values of the parameter, it is thus possible to obtain an estimation of how the solver scales along the dimension represented by the parameter. Unfortunately, the adequacy of the benchmarks proposed in the literature is seldom discussed. For instance, as we will see, some benchmarks have been reported to be solved efficiently by the proposed solvers, e.g., in linear time along some parameter for which no polynomial-time algorithm is known. When this is the case, the likely explanation is that the benchmarks do not represent the complexity of the problem. (The alternative being that the solver used in the experiments improves over the known complexity results.)

The sheer number of elements that contribute to the definition of the UAQ problem complicates the selection of the parameters. The elements characterizing the RBAC policy include the number of roles $|R|$, the number of permissions $|P|$, the number of DMER constraints $|C|$ as well as their specific features (e.g. $\widehat{rs}$ and $\widehat{t}$). One may even consider features of the PA relation, such as the maximum number of roles that contain any given permission (referred as $\widehat{R_P} = \max_{p \in P} |R_p|$ in Section 3). The components of the query also contribute to the complexity of the problem. These include the security objective (any, min, max), the number of requested permissions that *must* be granted, i.e. $|P_{lb}|$, and the number of requested permissions that *can* be granted, i.e. $|P_{ub}|$.

In previous work (see, e.g., [2, 9, 13, 18]), various benchmark problems parametric in any of these aspects have been put forward. To the best of our knowledge, the most extensive collection of parametric benchmarks so far is presented in [13] and summarized in Table 2. The benchmarks are parametric in $|R|$, $\widehat{R_P}$, $|D|$, $\widehat{rs}$, $\widehat{t}$, $|P_{lb}|$, $|P_{ub}|$, and *obj*. To illustrate, consider the benchmark problems named "roles" in the table. They are parametric in $|R|$ (with $|R|$ ranging from 25 to 200), have 500 permissions ($|P|$) with every permission being assigned to exactly 3 roles (and thus $\widehat{R_P} = 3$), and 10 DMER constraints ($|C|$); each DMER constraint contains 10 roles ($\widehat{rs}$) and the value of $\widehat{t}$ is set to 3. The cardinality of P_{lb} and P_{ub} are set to 7 and 20 respectively. Only the optimization objective min is considered.

While the benchmarks in [13] are a first attempt to provide a comprehensive evaluation along a number of significant dimensions, they still suffer from the following shortcomings:

(1) it is not always clear if and how these benchmarks represent the complexity of the UAQ problem;
(2) only the optimization objective min is considered, and they are therefore not suitable for evaluating the performance of the solvers when different optimization objectives, most notably max, are considered.

In order to overcome these limitations, we introduce a methodology that can be used both to evaluate existing benchmarks and to guide the design of new ones.

The complexity results introduced in Section 3 can be used to both validate and guide the design of benchmarks by leveraging the following observations:

- If a family of UAQ problems is known to be solvable in polynomial time, then by running a solver against this family of problems we can check how the performance of the solver compares with that of the known algorithms: if the time spent by the solver grows, e.g., exponentially as the size of the problem increases, than the solver is clearly inefficient against this family of problems.
- Dually, if a family of UAQ problems is known to be NP-hard, but the time spent by any solver has a polynomial growth as the size of the problem increases, then, assuming that $P \neq NP$, this means that the family of the problems (i.e. the benchmark) considered does not represent adequately the complexity of the problem.

The algorithms introduced in Section 3.1 play a key role in the proposed methodology. In fact, they provide upper bounds on the asymptotic growth rate for UAQ solvers which, to the best of our knowledge, are the best upper bounds currently available in the literature. Yet, it must be noted that the proposed methodology will yield different, improved results as soon as new complexity results will become available. Thus the benchmarks proposed and used in this paper could be improved consequently.

As already mentioned in Section 2, in our work we consider $P_{ub} = P$. We note that under this condition Algorithm 1 (asymptotically) outperforms Algorithm 2 if $obj = max$ over the benchmarks generated through our methodology. To see this it suffices to observe that $2^{\widehat{R_P}|P_{ub}|} < 2^{|R|}$ cannot possibly hold. In fact, $2^{\widehat{R_P}|P_{ub}|} < 2^{|R|}$ implies $\widehat{R_P}|P_{ub}| < |R|$ and, since $P_{ub} = P$, we have $\widehat{R_P}|P| < |R|$. Since during the benchmarks generation we will assign exactly $\widehat{R_P}$ roles to each permission, this would mean having far more roles than permissions. It is easy to see that in this case we would have many roles with no permission assigned, which cannot be the case. This means that any reasonably performing solver should never exhibit exponential growth over benchmarks parametric in $\widehat{R_P}$ or $|P_{ub}|$.

The methodology described above can be used to understand and validate the benchmarks presented in in Table 2. Since these benchmarks are designed for $obj = min$ and Algorithm 3 (asymptotically) outperforms Algorithm 2, in our methodology we consider only Algorithms 1 and 3. Comparing the complexity of Algorithm 1 ($O(2^{|R|})$) and Algorithm 3 ($O(\widehat{R_P}^{|P_{lb}|})$), it appears that Algorithm 3 (asymptotically) outperforms Algorithm 1 over all benchmarks. For example, over *roles*, $2^{|R|} = 3.4{\times}10^7..1.6{\times}10^{60}$, while $\widehat{R_P}^{|P_{lb}|} = 2187$, then $2^{|R|} > \widehat{R_P}^{|P_{lb}|}$. Similar considerations hold for the remaining benchmarks. We then observe that the value of $|P_{lb}|$ is fixed in all benchmarks except *plb*. Algorithm 3 is therefore insensitive to the value of the respective parameter (R for *roles*, $|C|$ for *d*, etc.) and so should be any reasonably efficient solver when applied to these problems. All benchmark problems but *plb* can thus be used to check whether UAQ solvers are as effective as Algorithm 3 as the value of the respective parameter increases. Benchmark *plb* is instead parametric in $|P_{lb}|$ and we therefore expect the solving time of Algorithm 3 (and of any other solver) to increase exponentially as $|P_{lb}|$ increases.

Driven by the our methodology, we propose two new families of parametric UAQ problems, one with obj = min and one with obj = max. The benchmarks with obj = min are summarized in Table 3:

- *Plb_bigR* and *Plb_smallR* are both parametric in $|P_{lb}|$. *Plb_bigR* is designed to stress test solvers for increasing values of $|P_{lb}|$: for large values of $|R|$ (here set to 200), the best known algorithm (Algorithm 3) is exponential in $|P_{lb}|$ and thus we expect any solver to exhibit the same behavior. *Plb_smallR* is instead designed to check the effectiveness of solvers: Algorithm 1 is in $O(2^{|R|})$ and thus we know that the problem can be solved efficiently for sufficiently small values of $|R|$ (here set to 10).
- *R_bigPlb* and *R_smallPlb* are both parametric in $|R|$ and are dual to *Plb_bigR* and *Plb_smallR* respectively. *R_bigPlb* is designed to be used to stress test solvers for increasing values of $|R|$: for large values of $|P_{lb}|$ (here set to 100), the best known algorithm (i.e. Algorithm 1) is exponential in $|R|$ and thus we expect any solver to exhibit the same behavior. *R_smallPlb* can be used to check the efficiency of solvers: Algorithm 3 is in $O(\widehat{R_P}^{|P_{lb}|})$ and thus we know that the problem can be solved efficiently for sufficiently small values of $|P_{lb}|$ (here set to 2).

Table 2: Parametric Benchmarks from [13]

Name	SAT	$\|R\|$	$\|P\|$	$\widehat{R_P}$	$\|C\|$	$\widehat{rs}$	$\widehat{t}$	$\|P_{lb}\|$	$\|P_{ub}\|$
roles	F	25..200	500	3	10	10	3	7	20
d	F	100	500	3	10..100	10	3	7	23
rolesPerConstr	F	300	1000	3	20	10..100	3	5	30
t	F	100	500	3	20	25	2..12	6	10
plb	F	100	500	3	10	10	3	1..11	$20 - \|P_{lb}\|$

Table 3: Benchmark specifications for obj = min

Name	$\|R\|$	$\|P_{ub}\|$	$\widehat{R_P}$	$\|C\|$	$\widehat{rs}$	$\widehat{t}$	$\|P_{lb}\|$
Plb_bigR	200	400	5	0	-	-	5..50
Plb_smallR	10	400	5	0	-	-	5..50
R_bigPlb	10..100	400	5	0	-	-	100
R_smallPlb	10..100	400	5	0	-	-	2
RPhat_bigPlb	200	400	2..12	0	-	-	10
RPhat_medPlb	200	400	2..12	0	-	-	4
RPhat_smallPlb	200	400	2..12	0	-	-	1
Pub	200	100..1000	5	50	8	3	10
C	200	400	5	10..100	8	3	10
rshat	200	400	5	10	5..50	3	10
that	1000	1000	1	50	20	2..8	10

- *RPhat_bigPlb*, *RPhat_medPlb*, *RPhat_smallPlb* are parametric in $\widehat{R_P}$ and can be used to check the effectiveness of solvers: for big values of $|R|$ (here set to 200) the best algorithm is Algorithm 3 which is in $O(\widehat{R_P}^{|P_{lb}|})$. We thus know that the problem can be solved efficiently for sufficiently small values of $|P_{lb}|$ (here set to 10, 4 and 1 respectively). Note that $|P_{lb}|$ is the degree of the polynomial and therefore the time spent by the solver may differ significantly (for the values of $|P_{lb}|$ considered) as $\widehat{R_P}$ increases.
- *Pub* is parametric in $|P_{ub}|$. As already noted in Section 3.1, Algorithm 2 should always be slower than Algorithm 3. As a consequence, $|P_{ub}|$ should have a moderate impact on the time spent by the solver. This benchmark can therefore be used to check the effectiveness of solvers.
- *C*, *rshat*, and *that*, are parametric in $|C|$, $\widehat{rs}$ and $\widehat{t}$ respectively. Since for $o_p = min$ these parameters do not contribute to the asymptotic complexity of any algorithm presented in Section 3.1, these benchmarks can be used to check the effectiveness of solvers.

The benchmarks with obj = max are summarized in Table 4:

- *R_bigCt* and *R_smallCt* are parametric in $|R|$. *R_bigCt* is designed to stress test solvers for increasing values of $|R|$: when $|C|$ and $\widehat{t}$ are such that $|C|\widehat{t}$ is sufficiently large (here set to 150), the best algorithm (Algorithm 1) is exponential in $|R|$ and thus we expect any solver to exhibit the same behavior. *R_smallCt* is instead designed to check the effectiveness of solvers: when $|C|$ and $\widehat{t}$ are such that $|C|\widehat{t}$ is sufficiently small (here set to 10), Algorithm 4 can efficiently solve the problems independently from the size of $|R|$ and any efficient solver should do the same.
- *Pub* is parametric in $|P_{ub}|$. Since for obj = max Algorithm 1 outperforms Algorithm 2, any efficient solver should solve these problems in time which is independent from $|P_{ub}|$. This benchmark can then be used to check the efficiency of solvers.
- *RPhat* is parametric in $\widehat{R_P}$ and is analogous to the previous case as this problems can be solved efficiently.
- *C_bigR* and *C_smallR* are parametric in $|C|$. *C_bigR* can be used to stress-test solvers for increasing values of $|C|$: for large values of $|R|$ (here set to 200), Algorithm 4 is to be preferred to Algorithm 1. Since Algorithm 4 is exponential in $|C|$, we expect solvers to exhibit the same behavior. *C_smallR* can be used to check the efficiency of solvers: Algorithm 1 is in $O(2^{|R|})$ and thus the problem can be solved efficiently for sufficiently small values of $|R|$ (here set to 10).
- *that_bigR* and *that_smallR*, parametric in $\widehat{t}$, are analogous to the previous case. We then expect that solvers exhibit exponential growth over *that_bigR*, while *that_smallR* can be solved efficiently.
- *rshat_smallCt*, *rshat_medCt* and *rshat_bigCt* are parametric in $\widehat{rs}$. For big values of $|R|$ (here set to 200), Algorithm4 outperforms Algorithm 1. Algorithm 4 is in $O(\widehat{rs}^{|C|\widehat{t}})$ and these problems can thus be solved efficiently for sufficiently small values of $|C|\widehat{t}$ (here set to 3). *rshat_smallCt* can then be used to check the efficiency of solvers. *rshat_medCt* and *rshat_bigCt* can be used to see how the values of $|C|\widehat{t}$ (here set to 9 and 30) affect the complexity of the problem.

Table 4: Benchmark specifications for $obj = \mathbf{max}$

Name	$\|R\|$	$\|P_{ub}\|$	$\widehat{R_P}$	$\|C\|$	$\widehat{rs}$	$\widehat{t}$	$\|P_{lb}\|$
R_bigCt	10..100	400	5	50	8	3	10
R_smallCt	10..100	400	5	5	3	2	10
Pub	200	100..1000	5	50	8	3	10
RPhat	200	400	20..200	50	8	3	10
C_bigR	200	400	5	10..100	8	3	10
C_smallR	10	400	5	10..100	8	3	10
that_bigR	1000	1000	1	50	20	2..12	10
that_smallR	20	400	5	10	12	2..12	10
rshat_bigCt	200	400	5	10	5..50	3	10
rshat_medCt	200	400	5	3	5..50	3	10
rshat_smallCt	200	400	5	1	5..50	3	10
Plb	200	400	5	20	5	2	5..50

- *Plb* is parametric in $|P_{lb}|$. Since for $o_p = max$ this parameter does not contribute to the asymptotic complexity of any algorithm presented in Section 3.1, this benchmark can be used to check the effectiveness of solvers.

Notice that we do not include benchmarks parametric in the "size" of the role hierarchy since, as already pointed out in Section 3, it does not contribute to the complexity of the UAQ problem. The benchmarks in Table 3 and in Table 4 are obtained by randomly generating a RBAC policy with the specified number of roles and permissions (ensuring that each permission is assigned to exactly $\widehat{R_P}$ roles), the specified number of DMER constraints of the form DMER($\{r_1, \ldots, r_m\}, t$) with $m = \widehat{rs}$, $t = \widehat{t}$ and a query with with P_{lb} with $|P_{lb}|$ randomly selected permissions and $P_{ub} = P$.

5 EXPERIMENTAL RESULTS

In Section 4, we presented an overview of the existing benchmarks and introduced a methodology to guide the design of benchmarks parametric with the relevant aspects of the UAQ problem. In this section, we first discuss the benchmarks presented in [13] and then experimentally compare the performance of the following solvers against those benchmarks and the ones presented in Table 3:

- 2D-Opt-Search [13]: a search-based solver leveraging the FPP result;
- 2D-Opt-CNF [13]: a SAT-based solver that leverages the reduction of the UAQ Decision Problem to SAT, zChaff [1] as SAT solver a state-of-the-art SAT solver and a two-dimensional binary search to solve UAQ problems;
- AQUA [2]: a SAT-based solver that employs a reduction of the UAQ Problem to PMaxSAT and employs any state-of-the-art PMaxSAT solver to tackle the problem. In the experiments presented in this paper we used the Loandra PMaxSAT solver [3].

All the solvers mentioned above support joint optimization of permissions and roles. However, since we focus on problems where the optimization objective is limited to permissions only, we disabled the role optimization in 2D-Opt-Search and 2D-Opt-CNF.

In the second set of experiments, we focus our attention to $obj = max$ case and evaluate the performance of AQUA against the benchmarks in Table 4.

For each benchmark problem we generated 10 instances and ran the solvers against them. All data points in our plots represent the median value of the time spent by the corresponding solver. The experiments have been conducted on a PC with 2 64-bit Intel Xeon CPU X7350 (8 core) @ 2.93GHz and 47 GB RAM running Linux (Ubuntu 16.04.5 LTS).

For all the benchmarks we set the timeout to 600 seconds, after which the solver execution is interrupted and the problem is labeled as "skipped". We omitted the timed-out solvers from the plots.

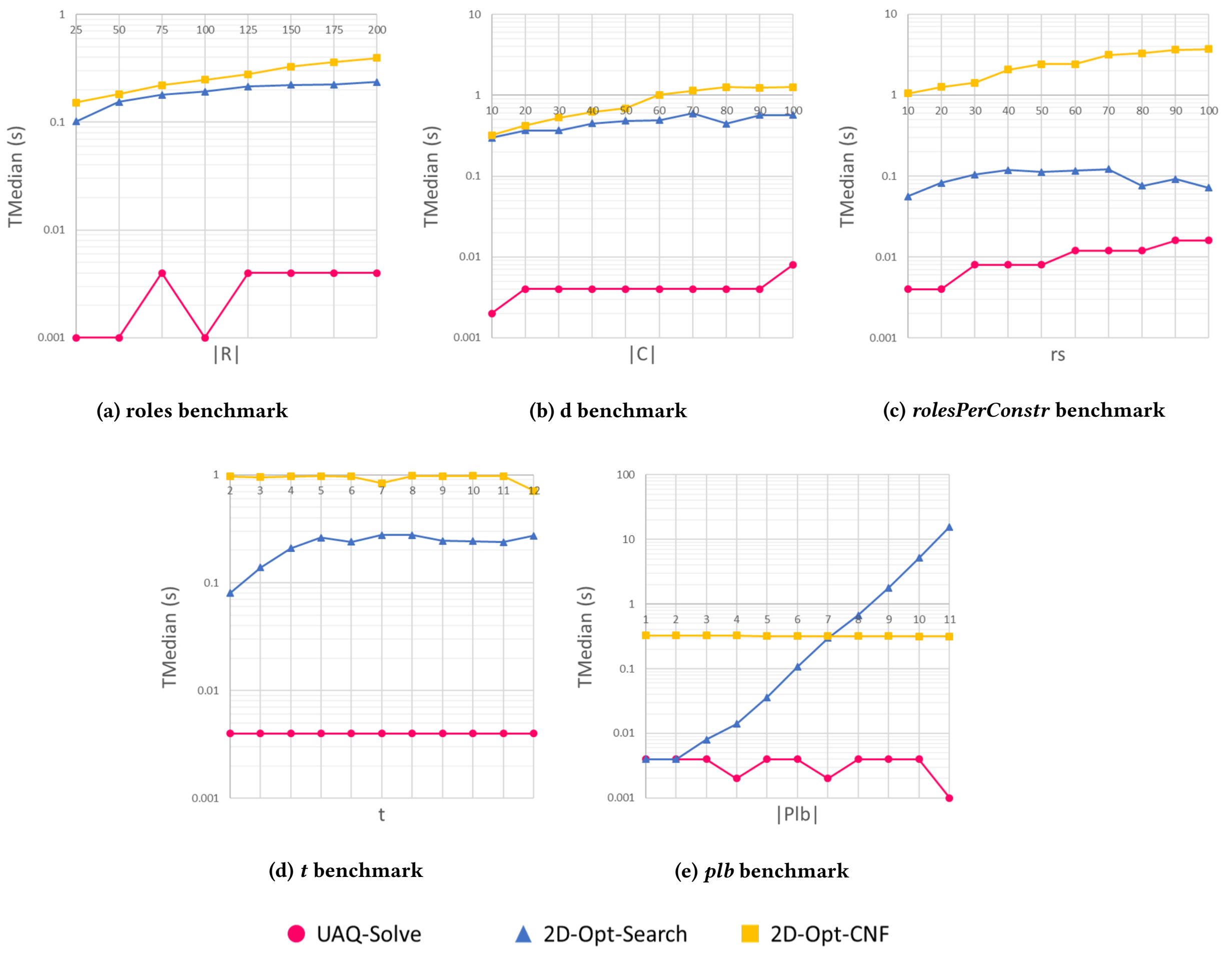
5.1 Benchmarks from [13]

Figure 1 presents the results of the evaluation of the existing benchmarks (cf. Table 2) and the results of the solvers' performance comparison against them.

Figure 1a shows the median solving time of the solvers over the *roles* benchmark. The value of $|R|$ ranges from 25 to 200, however the median solving time of AQUA never exceeds 4 milliseconds. This result confirms our prediction: in fact, the value of $\widehat{R_P}$ and $|P_{lb}|$ makes Algorithm 3 preferable to Algorithm 1.

Figures 1b, 1c, and 1d present the median solving time of the solvers over the benchmarks related to DMER constraints features, namely *d*, *rolesPerConstr*, and *t*. The sizes of $|C|$ and $\widehat{rs}$ vary from 10 to 100, and the size of $\widehat{t}$ varies from 2 to 12 in the respective benchmarks. AQUA exhibits a slight increase with the change of $|C|$ and $\widehat{rs}$, whose maximum values are 8 milliseconds and 16 milliseconds respectively. Instead, the median solving time is constant (4 milliseconds) when $\widehat{t}$ varies. These results also confirm our expectation: for $o_p = min$ the algorithms presented in Section 3.1 are insensitive to the variation of the parameters under discussion.

Finally, Figure 1e shows the experimental results obtained by running the solvers against the *plb* benchmark. Contrary to our expectation, AQUA could solve all the problems contained in the benchmark in a fraction of a second (at most in 4 milliseconds) even for very large values of $|P_{lb}|$, while it should have exhibited an exponential growth. This result means that the benchmark *plb*



(a) roles benchmark **(b) d benchmark** **(c) *rolesPerConstr* benchmark**

(d) *t* benchmark **(e) *plb* benchmark**

Figure 1: Performance of 2D-Opt-Search, 2D-Opt-CNF and AQUA on the benchmarks of Table 2

does not represent the complexity of the UAQ problem on varying numbers of $|P_{lb}|$.

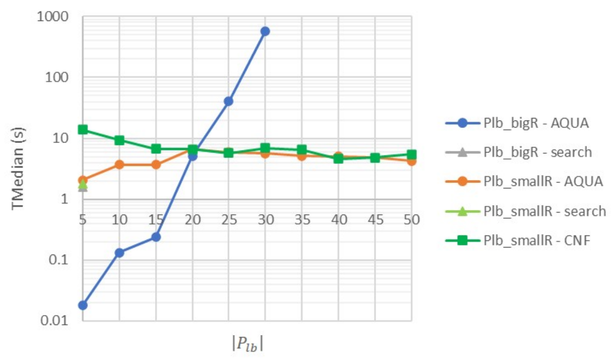
From the experiment, it follows that the benchmarks provided by [13] cannot be used to stress test solvers. They can nevertheless be used to evaluate their efficiency.

The results of the experiments in Figure 1 indicate that AQUA outperforms both 2D-Opt-Search and 2D-Opt-Search in most cases. Apart from *roles*, AQUA and 2D-Opt-CNF exhibit similar behavior, however there is always a distance of two orders of magnitude between the median solving time of the two solvers. Instead, the median solving time of 2D-Opt-CNF over *roles* slowly increases with the increase in $|R|$, while AQUA always takes 1 or 4 milliseconds. When running over the benchmarks *roles*, *d*, *rolesPerConstr*, and *t*, 2D-Opt-search always outperforms 2D-Opt-CNF. For *rolesPerConstr* the median solving time of 2D-Opt-search and 2D-Opt-CNF differs of at least one order of magnitude. For *plb*, 2D-Opt-CNF always solves the instances in 320 ms independently from the value of $|P_{lb}|$, whereas the time spent by 2D-Opt-Search has exponential growth, from 4 ms when $|P_{lb}| = 1$ to more than 15 sec when $|P_{lb}| = 11$.

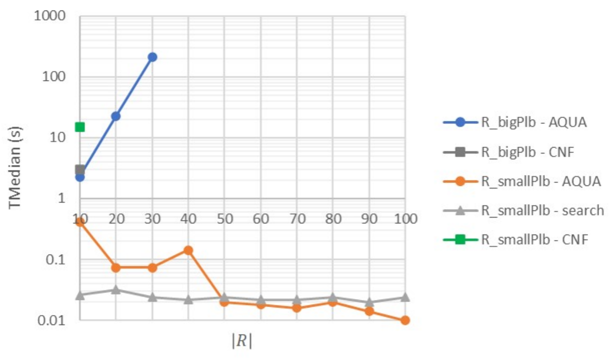
5.2 Our benchmarks with $obj = min$

We now turn our attention to our benchmark proposals where $obj = min$ (cf. Table 3). Figure 2 depicts the results of their experimental evaluation:

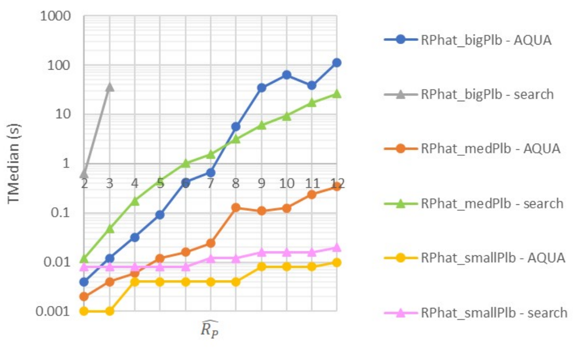
- The time spent by AQUA to solve *Plb_bigR* and *Plb_smallR* benchmarks meets the expectations. For *Plb_bigR*, the plot has a clear exponential growth, whereas for *Plb_smallR*, AQUA grows very slowly and even (slowly) decreases for $P_{lb} > 20$.
- Similar considerations hold for *R_bigPlb* and *R_smallPlb*.
- The results for *RPhat_bigPlb*, *RPhat_medPlb*, *RPhat_smallPlb* are of particular interest. As pointed out in Section 3, Algorithm 3 is in $O(\widehat{R_P}^{|P_{lb}|})$ and if $|P_{lb}|$ is bounded, then it can solve the problem in time which grows as a polynomial of degree $|P_{lb}|$. It can be noted that AQUA quickly solves all problems in *RPhat_smallPlb* (for which $|P_{lb}| = 1$), while the plot is considerably more steep for *RPhat_bigPlb* (for which $|P_{lb}| = 12$). This is consistent with the growth of polynomials of degree 1 and 12 respectively. The plot for *RPhat_smallPlb* (for which $|P_{lb}| = 4$) represents an intermediate situation.



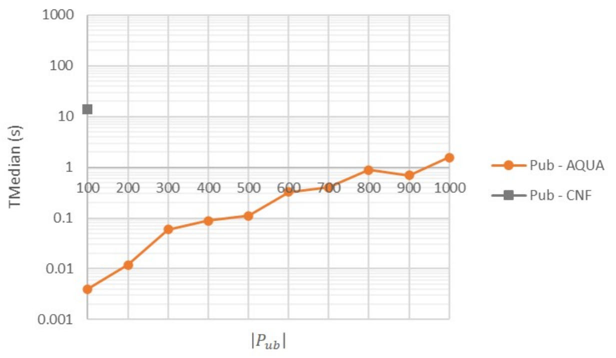
(a) Benchmarks parametric in $|P_{lb}|$



(b) Benchmarks parametric in $|R|$



(c) Benchmarks parametric in $\widehat{R_P}$



(d) Benchmark parametric in $|P_{ub}|$

Figure 2: Performance of 2D-Opt-Search, 2D-Opt-CNF and AQUA on the benchmarks of Table 3

- The plot for *Pub* indicates that the time spent by AQUA is not insensitive to $|P_{ub}|$. This is however not surprising if we consider that in our benchmarks $P_{ub} = P$ and the size of the encoding is in $O(|R||P|)$ and thus the formula grows linearly with $|P|$. Notice also that the time spent by AQUA is very small even for very large values of $|P_{ub}|$.
- Figure 2 does not show the results for the remaining benchmarks, namely *C*, *rshat*, and *that*, for reasons of space. However, as expected, AQUA quickly solves them. All instances contained in *C* and *t* are solved in around 0.1 seconds, while the median solving time over *rolesPerConstr* slightly increases from around 0.1 to 0.7 seconds with $\widehat{rs}$ ranging from 5 to 50.

The presented experimental results confirm that the benchmarks of Table 3 for problems with $obj = min$ is complete and satisfactory. It consists of hard benchmarks (*Plb_bigR* and *R_bigPlb*) which represent the complexity of the UAQ problem and can be used to stress test the solvers, and easy benchmarks (*Plb_smallR*, *R_smallPlb*, *RPhat_bigPlb*, *RPhat_medPlb*, *RPhat_smallPlb*, *Pub*, *C*, *rshat*, and *that*), which can be used to evaluate the solvers' performance . Figure 2 then also shows the results of the solvers' performance evaluation:

- The behavior of 2D-Opt-CNF on *Plb_smallR* is similar to that of AQUA, but when applied to *Plb_bigR* it reaches the timeout even for the smallest value of $|P_{lb}|$ (i.e. 5). 2D-Opt-Search can solve *Plb_smallR* and *Plb_bigR* only for the smallest value of $|P_{lb}|$.
- Similar considerations hold for R_bigR and R_smallR. Notice that 2D-Opt-Search performs remarkably well for R_smallR, but it reaches the timeout even for the smallest instance of R_bigR (i.e. 10).
- The behavior of 2D-Opt-Search on *RPhat_smallPlb* and *RPhat_medPlb* is similar to that of AQUA, but it reaches the timeout for *Plb_bigR* for a small value of $\widehat{R_P}$ (i.e. 2). Instead, 2D-Opt-CNF reaches the timeout for all the instances contained in the three benchmarks.
- 2D-Opt-CNF reaches the timeout for the smallest value of $|P_{ub}|$ (i.e. 100), whereas 2D-Opt-Search reaches the timeout even for that value.
- Over the remaining benchmarks, namely *C*, *rshat*, and *that*, 2D-Opt-CNF and 2D-Opt-Search reach the timeout even for the smallest values of the respective parameters.

We can then conclude that the approach used by AQUA seems to allow an important reduction of the solving time compared to 2D-Opt-Search and 2D-Opt-CNF over almost all the benchmarks of Table 3.

5.3 Our benchmarks with $obj = max$

Figure 3 presents the experimental evaluation of the benchmarks with the optimization objective set to max (cf. Table 4) obtained by running AQUA against them.

From the experimental results we can draw the following observations:

- The time spent by AQUA to solve *R_bigCt* and *R_smallCt* meets the expectations. For *R_bigCt* the plot has a clear exponential growth behaving like Algorithm 1, whereas for *R_smallCt* the plot is almost constant, behaving like Algorithm 4, whose complexity does not depend on $|R|$.
- The time spent by AQUA to solve *Pub* does not meet the expectations as the plot has an exponential growth. Further analysis on this family of problems could lead to improvements of existing solvers.
- As expected, AQUA quickly solves the instances contained in *RPhat*, behaving similarly to Algorithm 1, whose complexity does not depend on $\widehat{R_P}$.
- The time spent by AQUA to solve *C_bigR* and *C_smallR* meets the expectations. For *C_bigR* the plot has a clear exponential growth, whereas for *C_smallR* the solving time is around 100 ms independently from $|C|$.
- As expected, the median solving time of AQUA over *that_bigR* grows exponentially like Algorithm 4, while AQUA quickly solves the instances contained in *that_smallR*, behaving like Algorithm 1.
- The plot for *rshat_smallCt* show that the problems can be solved quickly as $\widehat{rs}$ increases as long as the value of $|C|\widehat{t}$ is sufficiently small (3 in this case). The plots for *rshat_medCt* and *rshat_bigCt* show the effect of larger values of $|C|\widehat{t}$ which is consistent with the growth of polynomials of degree 9 and 30 respectively.
- Figure 3 does not show the results over *Plb* for reasons of space. However, as expected, AQUA quickly solves the the benchmark, behaving like Algorithm 3, whose complexity does not depend on $|P_{lb}|$. All instances are solved in at most 4 ms.

From the evaluation, we then conclude that the benchmarks of Table 4 is complete and satisfactory for $obj = max$. In particular, *R_bigCt*, *C_bigR*, *that_bigR*, and *rshat_bigCt* are hard benchmarks representing the complexity of the UAQ problem and then can be used to stress test the solvers; on the contrary, *R_smallCt*, *Pub*, *RPhat*, *C_smallR*, *that_smallR*, *rshat_medCt*, *rshat_smallCt*, and *Plb* are easy benchmarks and can be used to evaluate the solvers' performance.

We do not show the results of the evaluation of the efficiency of 2D-Opt-Search and 2D-Opt-CNF: by assuming the "at least one role per permission", 2D-Opt-Search may lead to sub-optimal results, thus the comparison with AQUA would not be fair; instead, it is not clear if 2D-Opt-CNF can tackle problems with $obj = max$. In fact, we have evidence of many inconclusive solutions (neither satisfiable nor unsatisfiable).

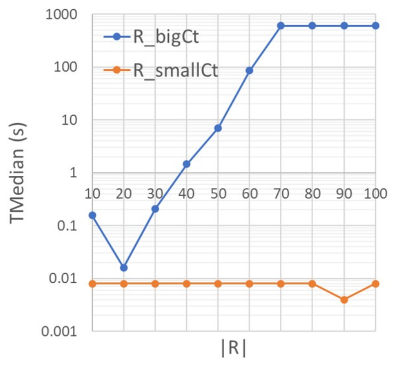
6 RELATED WORK

As UAQ is a central problem in RBAC systems, significant effort [9, 11, 12, 17] has been put to develop techniques for tackling it efficiently and to understand its underlying complexity. To our knowledge, [5] is the first paper that discusses UAQ where authors show that the complexity of finding minimal set of roles to be activated in a session that covers the permissions requested by the user is NP-complete. While they analyze UAQ in the presence of complex role hierarchies, they do not consider the constraint types (e.g. mutual exclusion of roles) available in RBAC. A natural extension of this work is presented in [20] where authors present two algorithms for solving UAQ problem instances. The first algorithm is a greedy search algorithm that looks for a set of roles covering the requested permissions while trying to minimize the additional permissions these roles provide. It is very efficient, but incomplete since it does not explore the space of all possible solutions. The second algorithm, aimed at providing completeness, is based on a simple generate-and-test strategy. It enumerates all subsets of roles assigned to the user until one is found that provides the needed permissions and satisfies all constraints. The problem with this algorithm is the first step may render the algorithm inefficient since it may need to generate $2^{|R|}$ solutions in the worst-case.

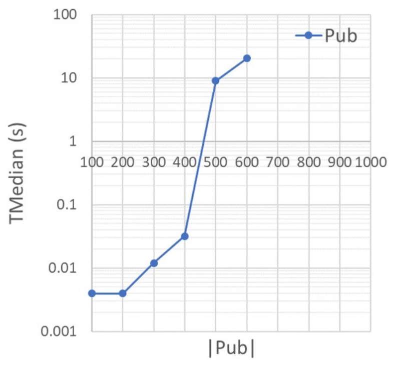
The more generic form for the UAQ problem where there are lower (P_{lb}) and upper (P_{ub}) bound permissions has been first proposed in [18]. The authors proposed two solutions for this problem. One is a variant of backtracking based search algorithm used in SAT solving, and the other is based on reducing the UAQ problem to MaxSAT. In general, the second solution performs better than the first one when obj = max or obj = min. The first approach is more efficient when an exact matching between the requested permissions and the available roles is sought. However, both approaches show very poor performance (as elaborately shown in [12]) when the number of roles increases.

The formal complexity analysis of different UAQ problem classes is also a point of strong scientific interest. For instance, [4] shows the complexities of UAQ problem by reducing it to a special case of set covering problem namely, container optimization. The reduction shows that the UAQ problem is NP-hard when obj = min, while it is in P when obj = max. However, the paper does not consider UAQ problems with dynamic constraints as we do in this paper. Moreover, these works did not either conduct any experimental analysis or employed extremely simple RBAC instances to evaluate their proposals. A general framework for the UAQ problem, that includes the optimization of number of roles as well as of number of permissions in proposed in [9] along with a comprehensive computational complexity analysis of various sub-cases and search-based solving techniques. However the proposed framework does not include DMER constraints which are one of the distinguishing feature of the UAQ problem considered in our work.

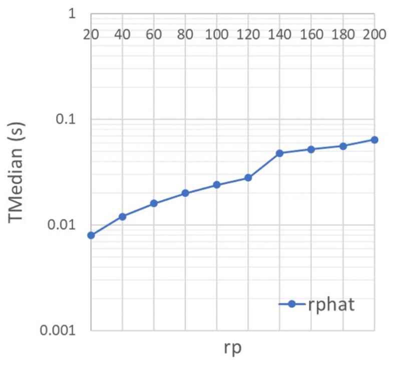
Mousavi et al. [13] show that there are two versions of the UAQ problem with constraints: the decision problem and the optimization problem. The authors also provide an extensive experimental analysis of the proposed techniques through a family of parametric benchmarks. At a high level, the decision version reduces the UAQ problem to SAT with an encoding similar to ours. The optimization version invokes a SAT solver in binary search while trying to minimize (or maximize) the set of roles that can be activated. They then present various algorithms (search-based or SAT reduction as discussed in Section 3.1) to tackle them efficiently. The authors also



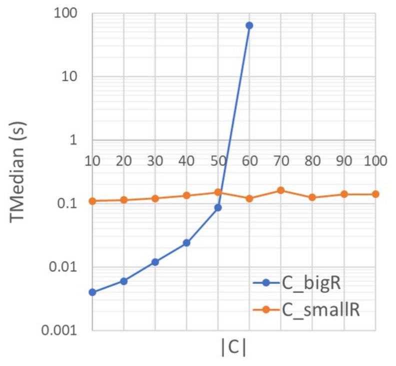
(a) Benchmarks parametric in $|R|$



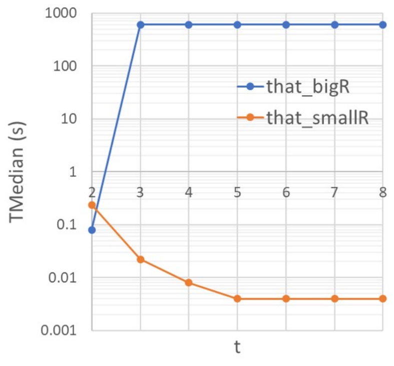
(b) Benchmark parametric in $|P_{ub}|$



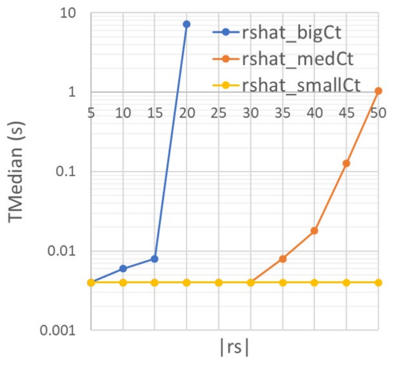
(c) Benchmark parametric in $\widehat{R_P}$



(d) Benchmarks parametric in $|C|$



(e) Benchmarks parametric in $\widehat{t}$



(f) Benchmarks parametric in $\widehat{rs}$

Figure 3: Performance of AQUA on the benchmarks of Table 4

show that the complexity result shown in [4] for the case $obj = \max$ is different when there are constraints in the UAQ instance. More specifically, they show that there is an upper bound (NP) for the general UAQ problem and the case $obj = \max$ is intractable if the UAQ instances have constraints. However, as we discussed in Section 4 in detail, our analysis revealed that the benchmarks they used in their experiments may not adequately reflect the complexities of various UAQ problem classes. An alternative approach, proposed again by Mousavi et al. [14], to the generation of benchmarks is to reduce the UAQ problem to constraint satisfaction problem (CSP) and employ a (hard) CSP instance technique [19]. This approach is particularly useful when generating hard UAQ problem instances, as the authors do, however they are hardly representative of the problem. In fact, our evaluations of their generation method mostly resulted with instances that are very easy to solve by a PMaxSAT solver.

More recently, a weighted variant of the UAQ problem has been proposed in [10] and [11] where the importance of a permission is also taken into consideration. The authors present various algorithms along with their complexity and compare them empirically. However, they consider role assignment (as opposed to role activation) as the primary constraint enforcement mechanism and the benchmarks used in their experiments is not systematic.

The research on the enforcement of security constraints, in particular Separation of Duty (SoD), in RBAC has also contributed to the discussion of UAQ. In [7] it has been observed that RBAC suffers from an under-specification problem due to the way “sessions” are employed in the standard [15]. The standard defines a set of high level functions to model the security requirements of applications while providing a bird-eye view to the authorization. However it fails in supporting some important principles, e.g. least privilege, for which run-time support becomes necessary. In [8] it is argued that that sessions are very useful for the dynamic management of roles. That this can be done efficiently is shown [2, 17]. More specifically, [2] shows that the definition of DMER constraints can be extended so to cover multiple sessions (i.e. MS-DMER) and role activation history (SS-HMER and MS-HMER). This way, the authorization constraints can be span multiple sessions and the the role activations occurred in the past.

7 CONCLUSIONS

In this paper, we presented a systematic overview of the computational complexity, existing algorithms and available benchmarks pertinent to the UAQ problem. Our analysis of the currently available benchmarks revealed that they are inadequate to analyze the inherent complexity of different UAQ problem classes. We then proposed a methodology to generate UAQ benchmarks starting from the known complexity results and used them to evaluate the state-of-the-art UAQ solvers. Our experimental results do not only show the effectiveness of the solvers over various UAQ problem classes but also the impact of the chosen parameter in the overall performance.

Since we focused on problems where the optimization objective is limited to permissions only, as future work we would like to investigate the generation for benchmarks of joint optimization. In addition, we would like to apply the methodology proposed in this paper to the UAQ problem specification framework introduced in [9], using the complexity results provided in that work to guide the generation of benchmarks.

REFERENCES

[1] [n.d.]. *zChaff.* http://www.princeton.edu/~chaff/zchaff.html

[2] Alessandro Armando, Silvio Ranise, Fatih Turkmen, and Bruno Crispo. 2012. Efficient run-time solving of RBAC user authorization queries: pushing the envelope. In *Second ACM Conference on Data and Application Security and Privacy (CODASPY).* 241–248.

[3] Jeremias Berg, Emir Demirovic, and Peter J. Stuckey. 2019. Core-Boosted Linear Search for Incomplete MaxSAT. In *Integration of Constraint Programming, Artificial Intelligence, and Operations Research - 16th International Conference, CPAIOR 2019, Thessaloniki, Greece, June 4-7, 2019, Proceedings.* 39–56.

[4] Liang Chen and Jason Crampton. 2009. Set covering problems in role-based access control. In *Proceedings of the 14th European conference on Research in computer security (ESORICS'09).* 689–704.

[5] S. Du and J. B. D. Joshi. 2006. Supporting authorization query and inter-domain role mapping in presence of hybrid role hierarchy. In *SACMAT.* 228–236.

[6] John Larusic and Abraham P. Punnen. 2014. The asymmetric bottleneck traveling salesman problem: Algorithms, complexity and empirical analysis. *Comput. Oper. Res.* 43 (2014), 20–35.

[7] Ninghui Li, Ji-Won Byun, and Elisa Bertino. 2007. A Critique of the ANSI Standard on Role-Based Access Control. *IEEE Security & Privacy* 5, 6 (2007), 41–49.

[8] N. Li, M. V. Tripunitara, and Z. Bizri. 2007. On mutually exclusive roles and separation-of-duty. *ACM Trans. Inf. Syst. Secur.* 10 (May 2007). Issue 2.

[9] Jianfeng Lu, James B. D. Joshi, Lei Jin, and Yiding Liu. 2015. Towards complexity analysis of User Authorization Query problem in RBAC. *Computers & Security* 48 (2015), 116–130.

[10] Jianfeng Lu, Zheng Wang, Dewu Xu, Changbing Tang, and Jianmin Han. 2017. Towards an Efficient Approximate Solution for the Weighted User Authorization Query Problem. *IEICE Transactions* 100-D, 8 (2017), 1762–1769.

[11] Jianfeng Lu, Yun Xin, Zhao Zhang, Hao Peng, and Jianmin Han. 2018. Supporting user authorization queries in RBAC systems by role-permission reassignment. *Future Generation Comp. Syst.* 88 (2018), 707–717.

[12] Nima Mousavi. 2014. *Algorithmic Problems in Access Control.* Ph.D. Dissertation. University of Waterloo, Canada.

[13] Nima Mousavi and Mahesh V. Tripunitara. 2012. Mitigating the Intractability of the User Authorization Query Problem in Role-Based Access Control (RBAC). In *NSS.* 516–529.

[14] Nima Mousavi and Mahesh V. Tripunitara. 2015. Hard Instances for Verification Problems in Access Control. In *Proceedings of the 20th ACM Symposium on Access Control Models and Technologies, Vienna, Austria, June 1-3, 2015.* 161–164.

[15] National Institute of Standards and Technology (NIST). 2004. Role-Based Access Control. *American National Standards Institute, Inc.* (2004).

[16] C. Sinz. 2005. Towards an Optimal CNF Encoding of Boolean Cardinality Constraints. In *Principles and Practice of Constraint Programming (CP).* 827–831.

[17] Fatih Turkmen. 2012. *Exploring Dynamic Constraint Enforcement and Efficiency in Access Control.* Ph.D. Dissertation. University of Trento, Italy.

[18] G. T. Wickramaarachchi, W. H. Qardaji, and N. Li. 2009. An efficient framework for user authorization queries in RBAC systems. In *SACMAT.* 23–32.

[19] Ke Xu, Frédéric Boussemart, Fred Hemery, and Christophe Lecoutre. [n.d.]. A Simple Model to Generate Hard Satisfiable Instances. In *IJCAI-05, Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence, Edinburgh, Scotland, UK, July 30 - August 5, 2005.* 337–342.

[20] Y. Zhang and J. B. D. Joshi. 2008. UAQ: a framework for user authorization query processing in RBAC extended with hybrid hierarchy and constraints. In *SACMAT.* 83–92.